2015 IEEE Symposium Series on Computational Intelligence

Evolving Robotic Neuro-Controllers Using Gene Expression Programming

J Mwaura
Department of Computer Science
University of Pretoria
Pretoria, South Africa
Email: jtmwaura@gmail.com

Ed Keedwell
Department of Computer Science
CEMPS,University of Exeter
Exeter, UK
Email: e.c.keedwell@exeter.ac.uk

***Abstract*—Current trends in evolutionary robotics (ER) involve training a neuro-controller using one of the various population based algorithms. The most popular technique is to learn the optimal weights for the neural network. There is only a limited research into techniques that can be used to fully encode a neural network (NN) and therefore evolve the architecture, weights and thresholds as well as learning rates. The research presented in this paper investigates how the chromosomes of the gene expression programming (GEP) algorithm can be used to evolve robotic neural controllers. The designed neuro-controllers are utilised in a robotic wall following problem. The ensuing results show that the GEP neural network (GEPNN) is a promising tool for use in evolutionary robotics.**

I. Introduction

The state of the art in evolutionary robotics (ER) involve the evolution of a full neural network or parts of a neural network using an evolutionary algorithm. This study is referred to as neuro-evolution or evolutionary neural networks (ENNs) [39]. The use of artificial neural networks(ANNs) in tandem with evolutionary algorithms (EAs) is normally justified by the following: ANNs are able to relate the input to the required outputs; in that case they can be used in predictions using historical data. They have also shown to be able to generalize between samples, in the way humans do [27], they also show 'graceful degradation' which means that removing one or more units results in reduced performance, not complete failure. They have also shown to be tolerant to noise in the data.

EAs are used to either evolve the architecture of the ANN, or to evolve weights, learning parameters or both. EAs are very suitable in learning the correct weights since they deal with a population of networks and no detailed specification of the network response is required [27], [38], [39]. When using EA for weights and learning parameters, the weights are directly encoded to the genotype either as a string of real values or string of binary values [27], [38], [39]. The main aim of evolving ANN weights is to find near optimal weights that can be used with a fixed ANN architecture to solve a given problem. In robotic neuro-evolution, genetic algorithms (GAs) have been used extensively to evolve suitable weights. Various work using GA to evolve weights include [9] [25] [23], [24] [37] [33].

The architecture of an ANN is made up by the weight connections and the transfer functions of each node in the network. Thus, a good network architecture is crucial to the success of solving a problem [39]. A key issue in evolving neural network architectures is to make a decision on how the architecture shall be encoded into an EA chromosome. Some work where the ANN architectures have been evolved for ER problems include [3], [2]. Other examples where neural network architectures have been evolved include [4], [5], [12], [13], [16], [27], [34], [35].

The evolution of connection weights and network architectures deal with the components of the ANN. The evolution of the learning rate or weight updating rule on the other hand, has to deal with the training of the neural network and hence to generate the dynamic behaviour displayed by the ANN [39]. EAs are suited to evolve learning rules or the learning parameters as they can discover unique parameters that lead to robust adaptability of ANN [31]. Examples of the evolution of learning rates in the ER domain can be found in [11], [30] [10], [36]. A comprehensive review can be found in [39].

The capability of the Gene Expression Programming (GEP) algorithm to evolve a neural network's architecture, weights and thresholds for such problems as XOR and 6-multiplexer problem has been investigated in [7]. However, whereas GEP algorithm has been used in ER domain, its capability to evolve robotic neural controllers has not been investigated. Consequently, the main objective of the work reported in this paper is to investigate the capabilities of GEP to evolve robotic neuro-controllers. The evolved controllers are used in a wall following robotic problem. The mechanism is referred to as GEP Neural Networks (GEPNN). The obtained results are compared to those of a multiple output GEP algorithm.

The rest of the paper is structured as follows: an introduction of GEP and multiple gene GEP is presented in section II. A description of how GEP can be used to evolve neural networks is presented in Section III. The experiment set up including the robot environments and algorithms implementations is presented in Section IV. In Section V, the results of the experiments are presented and a discussion follows. Section VI provides a conclusion for the paper.

II. Gene Expression Programming

Gene Expression Programming (GEP) is a population based evolutionary algorithm similar to genetic algorithms (GA) and genetic programming (GP). The algorithm is a variant of GP as it is used to evolve computer programs as well

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.153

IEEE computer society

as having the ramified tree structures commonly associated with GP. However, GEP genes are formed using fixed-length linear strings (similar to GA chromosomes) which are later expressed to variable length tree-like structures (similar to GP encoding). The fixed-length strings are the genotypes while the tree-structures, also referred to as expression trees (ETs), are the phenotypes.

Similar to GP, the GEP genes are formed using functions and terminals. The functions could include logical and conditional operators or any other user specified functions. The terminals, on the other hand, are the input variables to the GEP system. They could include mathematical symbols e.g. x, y or robotic end effectors (e.g. sensor encodings, motor settings etc).

The structure of a GEP gene is made up of a head and tail section. The head section is made up of functions and terminals while the tail section contains only terminals. To construct the GEP gene, the length of the head section, the function set and terminal set have to be provided. The fixed-length of the gene is given by

$$\begin{aligned} &GEP_l = h + t \\ &where \\ &\qquad t = h(n-1) \end{aligned} \tag{1}$$

the constant, h, is the length of the head section, t is the length of the tail section while n is the maximum arity in the function set.

In order to solve a problem, a population of GEP chromosomes[1] under-go variation using seven genetic operators. These operators are grouped into three, i.e., recombination, transposition and mutation. The recombination operators are; one point recombination, two point recombination and gene recombination. One point and two point recombination are identical to similar operators in GAs. The gene recombination operator is only used if the GEP chromosome is composed of more than one gene. The transposition operators select genetic materials within a gene and move it within the same gene. The insertion sequence (IS) transposition operator selects a transposon (alleles within the gene) and inserts it only at the head section (except the root position). The root insertion sequence (RIS) transposition operator is similar to the IS except that the first element of the transposon must belong to the function set. Once the RIS transposon is selected, it is then placed at the root of the gene. In both the IS and RIS operations, the tail section remains unchanged. Where the GEP chromosome is made up of more than one gene, the gene transposition involves selecting an entire gene (apart from the first gene) as a transposon and shifting it to the beginning of the chromosome. The mutation operator is similar to the GA mutation implementation. An in-depth discussion of the genetic operators can be found on [17], [6].

In order to determine how well the GEP chromosome estimates the solution of the given problem, the chromosome has to be decoded. The decoding process, referred to as translation in GEP terminology, is carried out using a breadth-first parsing scheme. The resulting expression is known as an open reading frame (ORF) and is expressed as a parse tree referred to as expression tree (ET). The ET is the derived solution and is subjected to function fitness evaluations.

GEP chromosomes composed of single genes have been shown to perform well on various problems. This problems include sequence induction problems [8], [28], robotic problems [17], rules identification [40] etc. The next section discusses how the multiple gene GEP chromosomes are constructed.

A. *Multigenic Gene Expression Programming*

One of the main characteristics of GEP is that its chromosomes can be formed with more than one gene. This is unlike GA and the standard GP, where there is no difference between a gene and a chromosome. In GEP, the difference is clear, i.e. similar to systems biology a chromosome can be composed of more than one gene. A GEP chromosome composed of more than one gene is referred to as multigenic GEP chromosome (mgGEP) while a chromosome with only one gene is referred to as unigenic GEP (ugGEP) chromosome.

In order to construct mgGEP chromosomes, the number of genes as well as the length of the head is chosen *a prior*. During the translation stage, each of the genes forms a sub-ET and the sub-ETs interact with each other to solve the given problem.

Genes in a mgGEP chromosome, can be combined using a special function, known as a linker, to form longer concatenated sequences. Alternatively, the genes may individually be utilised to solve a part of the problem. The evolved controllers (or partial controllers) are independent entities that work together to accomplish the set objectives. To make a clear distinction with these two mgGEP strategies, the concatenated GEP chromosomes, i.e, those using a linking function, are referred to as mgGEP chromosomes while the multiple output chromosomes are referred to as moGEP chromosomes. However, it is important to note that the main difference between these two multiple GEP algorithms is that the former utilises a linking function to combine the genes leading to only one output, while the latter leads to more than one output.

The capabilities of mgGEP chromosomes to evolve robotic behaviours has been shown on [19], [22]. In addition, modifications of the mgGEP algorithm to form varied length chromosomes has been shown by [1], [21]. Recently, moGEP chromosomes have been shown to be effective in evolving robotic controllers [20].

For a detailed discussion on GEP, please see [8], [22]. The next section discusses how the GEP genes can be used to encode a neural network.

III. Designing Neuro-Controllers Using GEP

ANNs consist of a set of processing elements, also known as neurons or nodes, which are interconnected. It can be described as a directed graph in which each node i performs a transfer function f_i of the form:

[1]GEP algorithm makes a clear distinction between a chromosome and a gene. A chromosome in GEP can be formed using more than one gene. Please see ensuing section for clarification.

$$y_i = f_i(\sum_{i=j}^{n} w_{i,j}x_j - \theta_i) \tag{2}$$

where y_i is the output of the node i, x_j is the jth input to the node, and $w_{i,j}$ is the connection weight between nodes i and j. θ_i is the threshold (or bias) of the node. Usually, f_i is non-linear, such as Heaviside, Sigmoid, or Gaussian function, n is the number of nodes. See [14], [26], [29], [32], [39].

To fully encode a neural network with the GEP chromosome, two new sections are added to the end of the gene [7]. These sections are the place holders for weights and thresholds. Using this mechanism, the GEP structure enables the algorithm to evolve the architecture of the neural network while the weights and threshold sections supplies the capability to learn the optimal weights and thresholds of the neural network.

In order to encode the weights, the length of the weight section is given by

$$W_s = h * n \tag{3}$$

while the length of the threshold section is equivalent to the length of the head, i.e. each function is associated with a threshold value. The weight section is added after the tail section while the threshold section is at the end of the gene. To make the GEPNN more elegant the weights and thresholds are placed into an array and the indexes of the array used in the gene representation. The weight array is generated within a particular range, for instance, the values could be $[-2, 2]$. The array length is also a user variable, for instance the array length could be 10 etc. Since the weights/thresholds are normally real numbers, the values could be held within the same array. When a weight/threshold is required during computation, then the value associated with the index is retrieved. This makes the GEPNN less cluttered and easier for a human reader. Nevertheless, implementation of the best way to represent this values can be left to the designer.

In order to fully comprehend how the GEP chromosome encodes a neural network (NN), consider a gene with $h = 3$, a function D whose arity, $n = 2$ and two terminals a, b. Using the provided information, the length of the weight section can be calculated as $W_s = 3 * 2 = 6$ and the length of threshold is 3, i.e. same as length of head section. Supposing we pick random weights and thresholds from the range $[0, 1]$, then the weights/threshold array could be

$$\{0.25, 0.425, 0.235, 0.32, -0.253, 0.82, 0.625, 0.123, 0.335\}$$

Using the given information a possible GEPNN encoding can be represented as

DDDbaba012387456

Figure 1 shows the ET expression of the ensuing GEPNN representation.

Once the GEP genome has been expressed, the ET representing the GEPNN can be derived. The weight values and threshold values can be added to the connection edges and each function has a bias value associated with it, the resulting ET is shown on Figure 2.

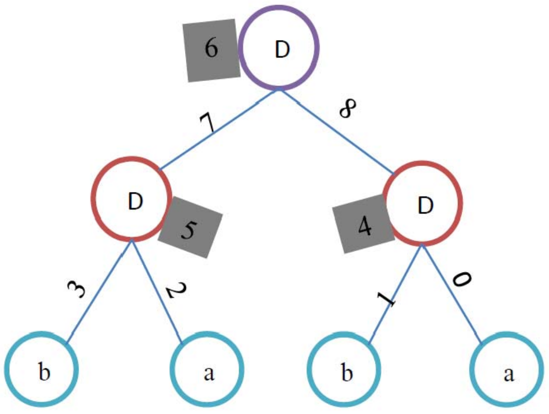


Fig. 1. A representation of a GEPNN expression tree.

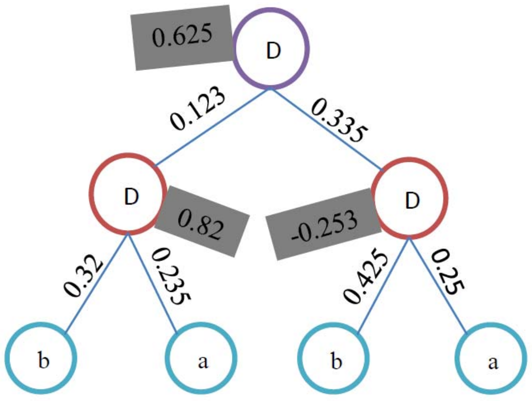


Fig. 2. An impression of a GEP expression tree with weights and bias added.

The derived GEPNN is equivalent to a two-input neural network with one hidden layer as shown by Figure 3.

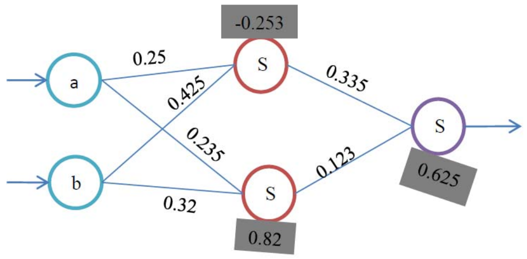


Fig. 3. GEP expression tree as a neural network

This representation shows that the GEP gene can be used to construct an ANN architecture containing the weights and threshold values. The next section discusses how the genetic operators are applied to the weight and threshold sections.

A. Genetic operators

In order to evolve the GEPNN, Ferreira [7] introduced specialised genetic operators to work on the weights and threshold sections. These extra specialised operators are

- Domain specific transposition: This operator works like the standard insertion sequence transposition operator. However, its operations is restricted to the neural network domains, i.e. weights and thresholds.
- Intragenic two point recombination: Similar to the transposition operator mentioned above, this recombinational operator selects materials only from the neural network domains. However, the view taken from this paper is that since this is a crossover operator, then there is no need for restriction as long as the GEP genes have fixed lengths.
- Direct mutation of weights and thresholds: This involves creating new weight values from the selected range and replacing the value selected for mutation. Mutation of weights and thresholds can be accomplished by having an array containing the weights (as proposed in Ferreira [7]) and then selecting a value from the array and mutating it. Alternatively, if one chooses not to create an array from where to select weight values, then weight mutation can be accomplished by randomly selecting weight/threshold value from the neural network domain and mutating it with a random value from the defined range.

B. GEPNN in machine learning

The initial paper introducing the concept of designing neural network using GEP, used the GEPNN to solve the XOR and 6-Multiplexer problems [7]. The GEPNN was shown to perform satisfactorily in finding the required solutions. Since the original paper, this technique has been used sparingly and only for classification problems. At the time of writing, the authors are not aware of any other existing approach to design robotic neuro-controllers using GEPNN.

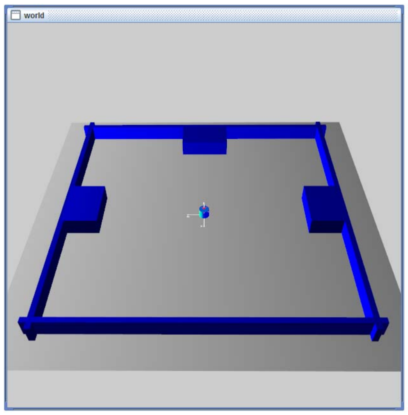
The motivation behind this work is that GEP algorithm has shown great promise in the evolution of robotic controllers [19], [20], [22]. As such the use of GEPNN in robotic neuro-evolution is a great step to showcase the power of GEP in the design of controllers. The presented work utilises GEPNN in a wall following robotic problem and compares the results with a controller evolved using the GEP tree structures.

The next section presents the experimental set up.

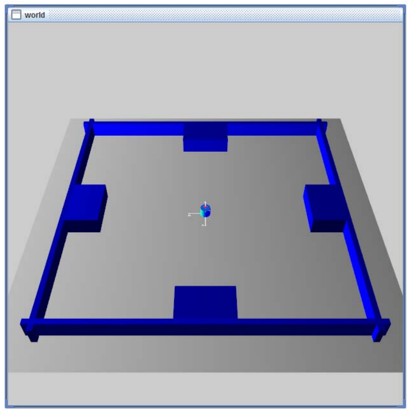
IV. Experimental set up

The robotic wall following problem was used to investigate the capability of GEP to evolve robotic neuro-controllers. In this problem, a robot tries to follow a wall and earns an objective fitness for visiting a new position next to the wall.

This section describes the experimental setup.



(a) Environment 1



(b) Environment 2

Fig. 4. Robot environments

A. Robot and environment implementation

For all described experiments, the *simbad* simulator[2] [15] was used to simulate the robot and its environment. To perform the required experiments, two types of environments were used as shown by Figure 4.

Table I shows the success predicates; maximum fitness achievable for each of the environments.

TABLE I. Success predicate

Environment	Optimal result
Environment 1	72
Environment 2	76

In all experiments, the simulated robot used eight infra-red sensors placed 0.25 radians apart along the perimeter of the top of the robot. Table II shows how the sensors were placed.

The robot sensors returned a value between $0.0m$ and $1.5m$. For the reported work, a sensor value equivalent to $0.0m$ meant that the robot was close to the wall. Conversely, a value of $1.5m$ meant that the robot was a minimum of $1.5m$ away from the obstacle. The robot movement was achieved using a simulated pair of wheels (left and right). In order to control the robot, simbad's differential drive kinematic model was used.

[2]http://simbad.sourceforge.net/

TABLE II. TERMINAL SET AND SENSOR POSITIONS

Sensor	Terminal Representations	Sensor positions
Front Sensor	F	0
Back Sensor	B	π
Left Sensor	L	$\frac{\pi}{2}$
Right Sensor	R	$\frac{3\pi}{2}$
Front Right Sensor	FR	$\frac{7\pi}{4}$
Back Right Sensor	BR	$\frac{5\pi}{4}$
Front Left Sensor	FL	$\frac{1\pi}{4}$
Back Left Sensor	BL	$\frac{3\pi}{4}$

In this kinematic model, the left and right wheel velocities are controlled independent of each other. The difference of the left and right wheel velocity produce an angular/rotational velocity while an equal output gives a linear velocity.

B. Algorithm parameters

The implemented GEP algorithm variants used the following attributes to describe the terminal and function sets.

1) Function set: For the GEPNN controllers, three summation functions were used. Similar to the standard ANN, the summation functions/units, sum the product of weight and the input value (robot sensor readings), i.e $\sum_{i=j}^{n} w_{i,j}x_j$, please refer to Equation 2. The summation functions were:

- D (double): This function had an arity $=2$ and therefore summed two values.
- T (Triple): This had an arity $=3$ and summed three input values.
- Q (quadruple): This had an arity $=4$ and hence summed four input values.

2) Terminal set: Table II shows the terminal representation used in the experiment. The value of the terminal was the sensor reading of corresponding robot sensors. No constants were used for the described experiments.

C. Activation functions for the GEPNN

The GEPNN was initially investigated using the following activation functions. These activation functions used the output of the algorithm summation functions, i.e. D, T, Q in their computation.

1) Linear function: The linear function is defined as

$$f(s-\theta) = (s-\theta) \tag{4}$$

where s is the weighted sum of the GEPNN input signals and θ is the bias. The output of this function is linear and is influenced by the weighted sum of the input signals and the bias value.

2) Ramp function: The ramp function was defined as

$$f(s-\theta) = \begin{cases} \lambda & \text{if } (s-\theta) \geq \epsilon; \\ -\lambda & \text{if } (s-\theta) \leq -\epsilon; \\ (s-\theta) & \text{otherwise.} \end{cases} \tag{5}$$

where λ is a scalar output value while ϵ is a threshold used to determine the function's output. In the experiments described here, λ was set to 2 while ϵ was set to 1.5. These values were not tuned for the problem. However, the maximum sensor reading, i.e. 1.5, was taken into consideration. The output of the activation function was thus in the range $(-2, 2)$.

3) Logistic Function: The logistic function was defined as

$$f(s-\theta) = \frac{1}{1+e^{(s-\theta)}} \tag{6}$$

where the constant, e, is the Euler's number. The output of this function was in the range $(0, 1)$

4) Hyperbolic tangent: The approximated hyperbolic tangent

$$f(s-\theta) = \frac{2}{1+e^{-1(s-\theta)}} - 1 \tag{7}$$

was used. The output of this activation function is in the range $(-1, 1)$

5) Evolving the GEPNN: Two genes each encoding a neural network were evolved. Each GEPNN gene gives an output for each of the robot wheels. This means that each robot wheel is controlled independently by an evolved GEPNN gene. Note that there was no linking function provided for the genes and that every GEPNN gene had an independent contribution to the overall robot behaviour. The fitness achieved was for the collective performance of the entire GEPNN chromosome (i.e. the two genes). In this case the robot was required to evolve a strategy to control its velocity and navigate around the wall perimeter.

The task of coordinating the wheel speeds is non-trivial since if one gene does not evolve a good solution or if it always returns a zero, the robot will continue rotating in the same place. The robot also has to achieve maximum speed when far from obstacles and while moving in a straight line. In addition, the robot has to decelerate when going around corners, as well as evolving the mechanism to start turning when protrusions are encountered.

6) Comparing the GEPNN with moGEP algorithm: For the purpose of comparison, the moGEP algorithm was used to evolve a controller for the wall following problem. For this experiment, the moGEP algorithm was used to evolve two sub-ETs. The output of each sub-ET is the sensor reading of that particular terminal symbol as shown in Table II. For instance, if the returned output is **F** with a *Front* sensor reading of $1.5m$, then this is translated as $1.5m/s$. Similar to the GEPNN controllers, the output of each of the genes was utilised to provide motor activity in m/s. All sensor readings are therefore utilised to provide a translation velocity for each of the two robot wheels. The output of the first sub-ET controlled the velocity of the left wheel while the output of the second sub-ET controlled the right wheel. The difference of the two outputs in terms of sensor readings produced an angular velocity while an equal output gave a linear velocity. The robot, thus, had to establish a link between linear and angular velocity in order to accomplish the wall following behaviour.

As can be deduced, the moGEP architecture is therefore similar to the GEPNN structure. As such, both these techniques utilised the differential drive kinematic model to control the robot movements, i.e, the left and right wheel velocities were controlled independent of each other during the course of the simulation.

The main difference with these two models is that each moGEP gene uses the output of the GEP ramified tree, whereas the GEPNN genes encode a neural network. Previous study [20] has shown that moGEP is a sufficient method to evolve robotic behaviours.

7) Genetic operators and algorithm parameter values: All the GEP operators were used in the experiment with the probabilities set as per Table III

TABLE III. PARAMETER SETTINGS

Parameters	**GEPNN**	**moGEP**
Maximum generations	500	500
Population size	100	100
Head size	4	4
No. of genes	2	2
Functions(D,Q,T)	3	$-$
Functions(IFLTE)	$-$	1
Weights/Threshold range	$(-2, 2)$	$-$
Weights array size	100	$-$
Terminals(see Table II)	8	8
Mutation probability	$\leftarrow 2/\text{chromosome size}$	$\rightarrow$
1-Point Recombination Probability	0.7	0.7
2-Point Recombination Probability	0.2	0.2
Weights/Threshold IS transposition	0.1	$-$
IS Transposition probability	0.1	0.1
RIS Transposition probability	0.1	0.1
Gene transposition probability	0.1	0.1
Gene recombination probability	0.1	0.1

The values of genetic operators probabilities, shown on Table III, are derived from work carried out by the authors in [18].

The implemented algorithms were generational and used replication to pass the best organism in the previous generation. Roulette wheel selection was used to select parent organisms. The algorithms were evaluated over 20 randomly seeded runs. Each controller in the population was allowed to control the robot for 50 virtual seconds. The rest of the parameters were set as defined in Table III.

8) Fitness function: The fitness was calculated as the summation of all new coordinates that a robot visits less any collisions. The robot was also penalised for remaining stationary. The following fitness function was used.

$$fitness = (\sum_{i=1}^{m} p_i(x_i, y_i)) - C - Wp \tag{8}$$

where the location (x_i, y_i) is a cell adjacent to the wall. The fitness point, $\mathrm{p_i}(x_i, y_i) = 1$ whenever (x_i, y_i) has not previously been visited and 0 otherwise. The collision penalty *C*, was set to 5 and the wandering penalty, *Wp*, was set to 10^3. The collision penalty was incurred when the intended movement would have led to the robot moving into an occupied cell (i.e. wall) whereas a wandering penalty was incurred when the robot visited any empty cells that were not adjacent to the wall. The total number of cells next to the wall, *m*, is adjusted before a run depending on which room type being used in the experiment. See Table I for the maximum number of cells next to the wall and the maximum fitness achieved in each environment.

[3]No tuning was carried out to determine optimal values for collision and/or wandering penalties.

V. RESULTS AND DISCUSSION

This section provides the results of the experiments described above. The main aim is to investigate the capability of the GEPNN to evolve controllers for a wall following robot. A comparison of the GEPNN performance is made using the moGEP algorithm.

Figure 5 shows the results of the GEPNN (with all the different activation functions) and the performance of the moGEP controllers. Table IV, shows the performance in terms of the percentage success rate. The success rate is defined as number of successful runs (where the maximum fitness was achieved) over the total number of runs, i.e. the 20 independent runs.

TABLE IV. %SUCCESS RATES

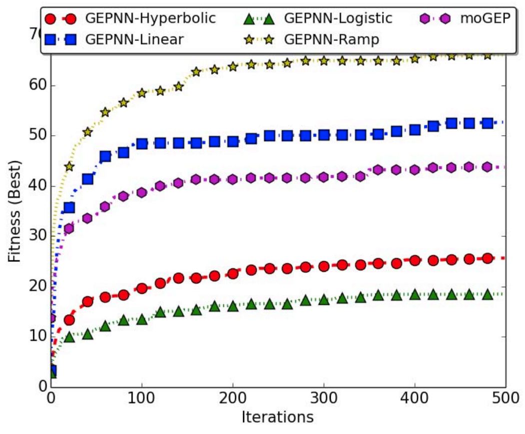
	GEPNN-Hyperbolic	**GEPNN-Linear**	**GEPNN-Logistic**	**GEPNN-Ramp**	**moGEP**
Env 1	0.0	20.0	0.0	50.0	0.0
Env 2	0.0	35.0	0.0	45.0	0.0

The performance results, Figure 5 shows that overall, the GEPNN with Ramp activation function performed better than all the other strategies in both problems. These results also show that the GEPNN with linear activation outperformed the moGEP algorithm in both problem cases. The GEPNN using logistic and hyperbolic functions had the lowest performances over the course of the algorithm run. The performance with various activation functions, notwithstanding here, these results show that GEPNN is a viable mechanism to evolve robotic neural controllers. The results shows that GEPNN (Ramp activation function) as more viable technique as it was more likely to achieve best fitness over the course of the run. The success rates, Table IV also show that GEPNN-Ramp had the best performance over the two problems.

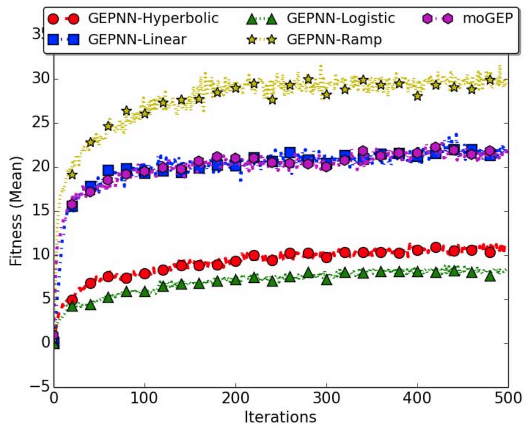
The good performance of the GEPNN-Ramp can be attributed by the λ values associated with the Ramp function. Equation 5 shows that the values of this function is scaled in the range $[-2, 2]$. This means that the robot could move at a speed of $2.0m/s$ when not near obstacles as well as reduce speed when near obstacles. The ramp function is a combination of the step and linear function. Since, the ramp has lower and higher boundaries, the robot's wheel velocity is more defined and hence the robot is able to navigate the environment more easily. For the linear function, the output is not defined, the wheel velocity can thus be as high as the GEPNN output. This means that whereas the robot is able to move very fast, it becomes difficult while trying to navigate through the protrusions hence the lower performance and success rate. For the logistic activation function, the values are scaled in the range $[0, 1]$ while the values of the hyperbolic function are scaled within $[-1, 1]$. This lower scaling means that it was likely that the robot moved at a slower speed and may not have covered similar distance as the robot using a GEPNN-Ramp controller.

A. Scaling the output of the activation functions

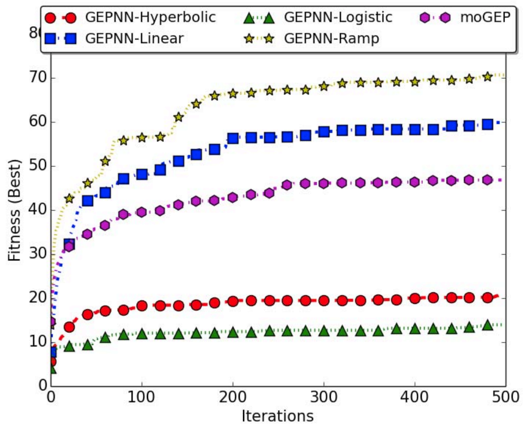
The previous section should that the performance of the GEPNN with various activation functions, could have been affected by the scaled values for the activation functions. In this section, all the activation functions are scaled to the



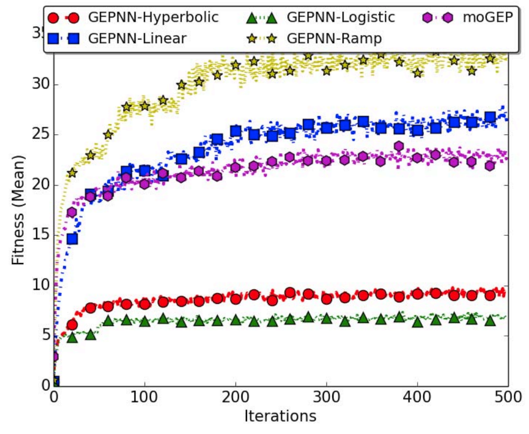
(a) Best Fitness in Environment 1



(b) Average Fitness Environment 1



(c) Best Fitness in Environment 2



(d) Average Fitness Environment 2

Fig. 5. Mean Fitness values over the iterations in the robot environments under investigation.

same range in order to allow a fair comparison. The min-max normalisation was used for the scaling. The min-max was defined as

$$\grave{v} = \frac{v - (v_{min})}{(v_{max} - v_{min})} * (\grave{v}_{max} - \grave{v}_{min}) + 1 \tag{9}$$

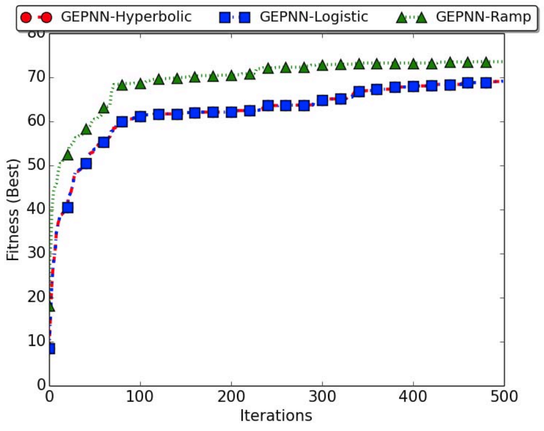
where $\grave{v}$ is the new output after normalisation, v is the value obtained from the activation function, v_{max} and v_{min} are the respective maximum and minimum values that the activation function can yield, and $\grave{v}_{max}$ and $\grave{v}_{min}$ is the new respective maximum and minimum values, i.e. the range.

The normalisation formula requires the use of expected minimum and maximum values. Since the linear activation function does not use minimum and maximum values, the linear function was not further used in the experiments. Figure 6 shows the average best fitness and mean fitness over the simulation runs, using the scaled activation function values.

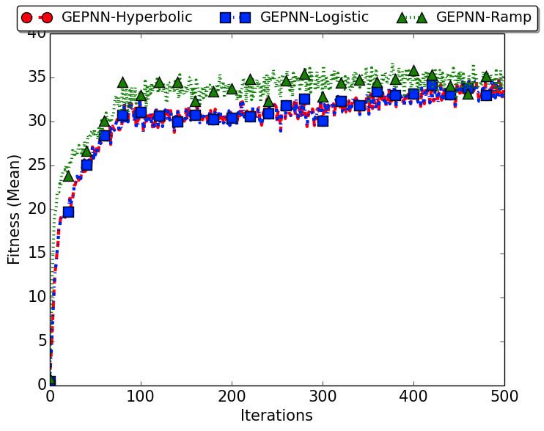
Results shown on Figure 6 suggest that scaling was critical to the performance of the GEPNN-Hyperbolic and GEPNN-Logistic. Since the output of the activation functions affect the wheel velocities, it is important to scale the output to the desired maximum and minimum speed in order to achieve optimal performance. As the results show, scaling led to a significant improvement of the performances of both the GEPNN-Hyperbolic and GEPNN-Logistic. For the GEPNN-Ramp, the values were already scaled within the range $[-2, 2]$ so the min-max scaling did not overly influence the results.

To further investigate whether scaling the output of the activation functions used for the GEPNN is required, a new set of experiments were carried out. In these new experiments, the set of functions of the GEPNN were implemented with each containing different activation functions. For instance, the summation function D, four functions were created, $D - Ramp$, $D - Hyperbolic$, $D - Linear$, $D - Logistic$ etc. This was carried out for all three summation functions. In total, there were 12 functions used for the GEPNN construction. The GEPNN was evolved with no scaling of the activations functions and the results were compared with a GEPNN-with scaling of the 12 functions and also with GEPNN-Ramp (scaled). Figure 7 shows the results when the GEPNN was run on Environment 1 and 2. Figure 8 shows the ranking of the algorithms using the Friedman's test.

Results shown on Figure 7 suggest that combining all

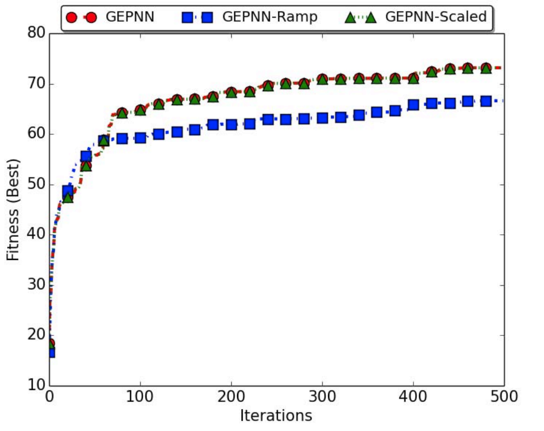


(a) Best Fitness in Environment 2 with scaled output

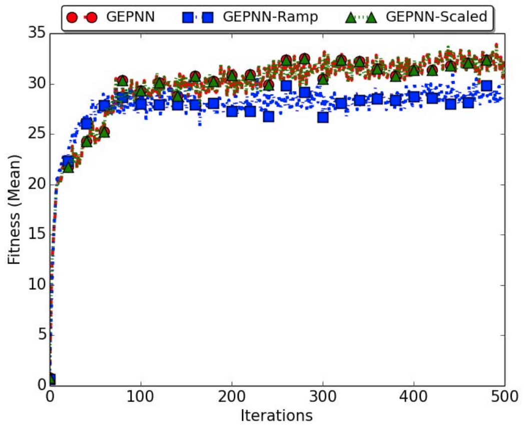


(b) Average Fitness Environment 2 with scaled output

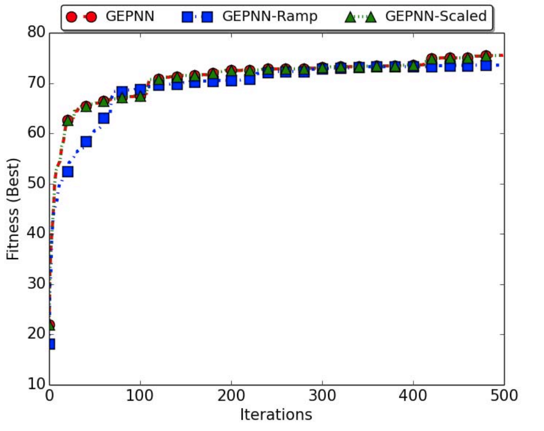
Fig. 6. Fitness values over the iterations in Environment 2 with min-max scaling of the activation function outputs



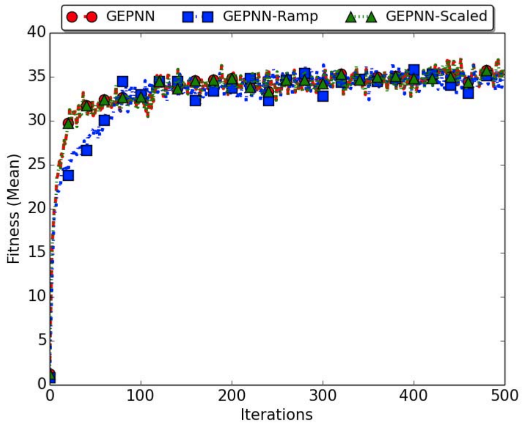
(a) GEPNN comparison Best Fitness in Environment 1



(b) GEPNN comparison Average Fitness in Environment 1



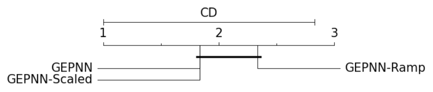
(c) GEPNN comparison Best Fitness in Environment 2



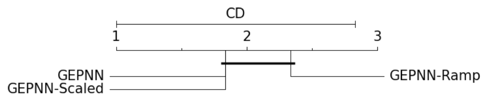
(d) GEPNN comparison Average Fitness in Environment 2

Fig. 7. GEPNN fitness values comparison with scaled activation functions output over the number of iterations

activation functions (and not scaling the outputs) performed equally with the GEPNN where all activation functions were used and output scaled. The two GEPNN strategies that used all activation functions outperformed GEPNN-with scaled



(a) Ranking and statistical using best fitness



(b) Ranking and statistical significance using average fitness

Fig. 8. Algorithm ranking and statistical significance testing using Friedman's test

Ramp in Environment 1, but performed equally in Environment 2. The algorithm ranking, Figure 8, shows that the GEPNN strategies using all activation functions were ranked first. However the Friedman's test also show that there was no statistical significance between the three strategies (The bold line below the ranks, Figure 8, means that there was no statistical significance in these results).

These results are significant in the context that they show that while designing robotic neuro-controllers, scaling of the output of activation functions is not required so long as all possible activation functions are included in the GEP function sets.

VI. Conclusion

This paper investigates how the GEP algorithm can be modified to evolve robotic neuro-controllers. Whereas the idea of designing neural networks is not new, this is the first work to investigate the capabilities of GEP neural nets to evolve robotic neural controllers. The paper discussed GEP as an algorithm and then reviewed the mechanism for evolving neural networks using the algorithm. Further, experiments are conducted using various activation functions. The GEPNN is used to evolve robotic neuro-controllers for two robotic environments. The ensuing results and discussion show that GEPNN is a viable technique for use in robotic neuro-evolution.

The current state of the art in ER involves the use of a neural network for robotic control. There is, therefore, a need to investigate more mechanism that can be used to design both the architecture of the neural network as well as learn the optima weights and threshold values and learning rates. Results obtained from this research, though of a preliminary nature, show that GEPNN is a promising algorithm for consideration in the ER domain.

The future focus of this work is to use GEPNN in more complex robotic problems and compare its performance with that of a neural network trained using particle swarm optimisation (PSO) and GAs. There is also a need to investigate what deep learning in artificial neural network would mean for a GEPNN and whether the use of multigenic GEP can entail deep learning.

Acknowledgment

The authors would like to thank Dr Michal Chudy (Graduate School of Technology Management) for the help with computing resources. This research was supported through the National Research Foundation of South Africa, under grant number: 89630.

References

[1] E. Bautu, A. Bautu, and H. Luchian, "AdaGEP-An Adaptive Gene Expression Programming Algorithm," in *Proceedings of Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2008, pp. 403–406.

[2] J. Blynel and D. Floreano, "Exploring the T-Maze: Evolving Learning-Like Robot Behaviours Using CTRNNs," in *Proceedings of the 2003 international conference on Applications of evolutionary computing*. Springer-Verlag, 2003, pp. 593–604.

[3] G. Capi and K. Doya, "Evolution of Neural Architecture Fitting Environmental Dynamics," *J. International Society for Adaptive Behaviour*, vol. 13, no. 1, pp. 53–66, 2005.

[4] D. Cliff, P. Husbands, and I. Harvey, "Explorations in Evolutionary Robotics," *J. Adaptive Behaviour*, vol. 2, pp. 73–110, 1993.

[5] ——, "Seeing the Light: Artificial Evolution, Real Vision," *From Animals to Animats 3, Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, pp. 392–401, 1994.

[6] C. Ferreira, "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems," *J. Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[7] ——, "Designing Neural Networks Using Gene Expression Programming," in *Proceedings of the 9^{th} Online World Conference on Soft Computing in Industrial Applications*, ser. Applied Soft Computing Technologies: The Challenge of Complexity, A. Abraham, B. De Baets, and B. Nickolay, Eds. Springer-Verlag, 2006, pp. 517–536.

[8] ——, *Gene Expression Programming: Mathematical Modelling by an Artificial Intelligence (2nd edition)*. Springer, 2006.

[9] D. Floreano and F. Mondada, "Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot," in *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA, 1994.

[10] ——, "Evolution of Plastic Neurocontrollers for Situated Agents," in *From Animals to Animats 4, Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB'1996)*, P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S. Wilson, Eds. MA: MIT Press, 1996, pp. 402–410.

[11] D. Floreano and J. Urzelai, "Evolutionary Robots with On-line Self-Organization and Behavioural Fitness," *J. Neural Networks*, vol. 13, pp. 431–443, 2000.

[12] I. Harvey, P. Husbands, and D. Cliff, "Issues in Evolutionary Robotics," in *Proceedings of the 3^{rd} International Conference on Simulation of Adaptive Behavior*, J.-A. Meyer, H. Roitblat, and S. Wilson, Eds., vol. 2, 1993, pp. 73–110.

[13] I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi, "Evolutionary Robotics: The Sussex Approach," *J. Robotics and Autonomous Systems*, vol. 20, pp. 205–224, 1997.

[14] S. Haykin, *Neural Networks*, 2nd ed. Prentice-Hall, 1999.

[15] L. Hugues and N. Bredeche, "Simbad: An Autonomous Robot Simulation Package for Education and Research," in *Simulation of Adaptive Behaviour*, 2007.

[16] P. Husbands, I. Harvey, D. Cliff, and G. Miller, "Artificial Evolution: A New Path for Artificial Intelligence?" *J. Brain and Cognition*, vol. 34, no. 1, pp. 130–159, 1997.

[17] J. Mwaura, "Evolution of Robotic Behaviour Using Gene Expression Programming," Ph.D. dissertation, University of Exeter, 2011.

[18] J. Mwaura and E. Keedwell, "Adaptive Gene Expression Programming Using a Simple Feedback Heuristic," in *Proceedings of the AISB*, Edinburgh, UK, 2009.

[19] ——, "Evolution of Robotic Behaviours Using Gene Expression Programming," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2010)*, Barcelona, Spain, 2010, pp. 1–8.

[20] ——, "On using gene expression programming to evolve multiple output robot controllers," in *2014 IEEE International Conference on Evolvable Systems, ICES*, 2014, pp. 173–180.

[21] J. Mwaura, E. Keedwell, and A. Engelbrecht, "Evolved Linker Gene Expression Programming: A new technique for Smbolic Regression," in *Proceedings of the 1st BRICS conference on computational Intelligence*, Porto de Galinhas, Brazil, 2013, pp. 1–8.

[22] J. Mwaura and E. Keedwell, "Evolving robot sub-behaviour modules using Gene Expression Programming," *Genetic Programming and Evolvable Machines*, pp. 1–37, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10710-014-9229-x

[23] A. L. Nelson, E. Grant, G. J. Barlow, and T. C. Henderson, "A Colony of Robots Using Vision Sensing and Evolved Neural Controllers," in *Proceedings of the IEEE/RSJ International Conference On Intelligent Robots And Systems*, vol. 3, 2003, pp. 2273–2278.

[24] A. L. Nelson, E. Grant, J. M. Galeotti, and S. Rhody, "Maze Exploration Behaviours Using an Integrated Evolutionary Robotics Environment," *J. Robotics and Autonomous Systems*, vol. 46, no. 3, pp. 159–173, 2004.

[25] R. Neruda and S. Slusny, "Evolving Neural Network which Control a Robotic Agent," in *Proceedings of the IEEE Congress on Evolutionary Computationn*, 2007.

[26] N. J. Nilsson, "Introduction to Machine Learning: An Early Draft of a Proposed Textbook." 1996.

[27] S. Nolfi and D. Floreano, *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, 2000.

[28] M. Oltean and C. Grosan, "A Comparison of Several Linear Genetic Programming Techniques," *J. Complex Systems*, vol. 14, pp. 285–313, 2003.

[29] D. Patterson, *Artificial Neural Networks: Theory and Applications*, 1st ed. Prentice Hall PTR, 1998.

[30] A. Radi and R. Poli, "Discovering Efficient Learning Rules for Feedforward Neural Networks Using Genetic Programming," in *Recent Advances in Intelligent Paradigms and Applications*, ser. Studies in Computational Intelligence, A. Abraham, L. Jain, and J. Kacprzyk, Eds. Springer Berlin / Heidelberg, 2003, pp. 133–159.

[31] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving Adaptive Neural Networks With and Without Adaptive Synapses," in *Proceedings of the 2003 Congress on Evolutionary Computation*, 2003.

[32] W.-H. Steeb, *The Non-Linear Workbook (3rd Ed)*. World Scientific Publishing Co. Pte, 2005.

[33] T. Thompson and J. Levine, "Scaling-up Behaviours in EvoTanks: Applying Subsumption Principles to Artificial Neural Networks," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[34] E. Tuci, M. Quinn, and I. Harvey, "An Evolutionary Ecological Approach to The Study of Learning Behaviour Using a Robot-Based Model," *J. Adaptive Behaviour*, vol. 10, pp. 201–221, 2002.

[35] J. Urzelai and D. Floreano, "Incremental Evolution with Minimal Resources," in *Proceedings of the International KHEPERA Workshop*, 1999.

[36] ——, "Evolution of Adaptive Synapses: Robots with Fast Adaptive Behaviour in New Environments," *J. Evolutionary Computation*, vol. 9, pp. 495–524, 2001.

[37] N. van Hoorn, J. Togelius, and J. Schmidhuber, "Hierarchical Controller Learning in a First-Person Shooter," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009, pp. 294–301.

[38] D. Whitley, "Genetic Algorithms and Neural Networks," *J. Genetic Algorithms in Engineering and Computer Science*, pp. 191–201, 1995.

[39] X. Yao, "Evolving Artificial Neural Networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[40] J. Zhong, L. Luo, W. Cai, and M. Lees, "Automatic Rule Identification for Agent-Based Crowd Models Through Gene Expression Programming," in *Proceedings of the 13^{th} International Conference on Autonomous Agents and Multiagent Systems (AAMAS2014)*, A. Lomuscio, P. Scerri, A. Bazzan, and M. Huhns, Eds., 2014, pp. 1125–1132.